Práctica 9: Hilos en Java

75.59 - Técnicas de Programación Concurrente I

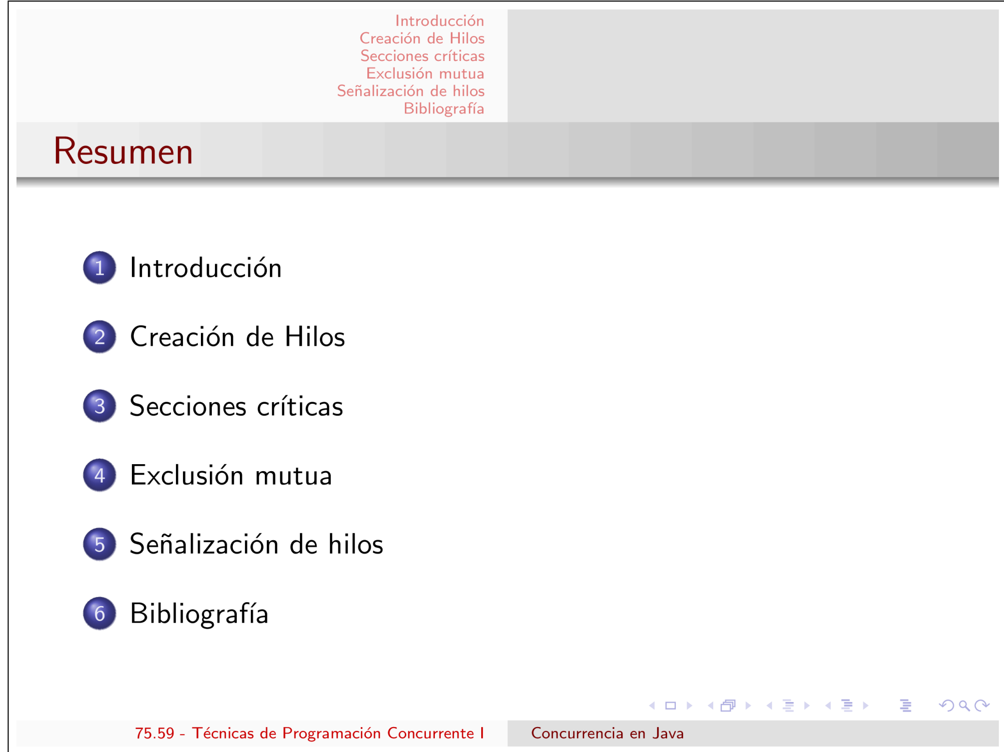
Ejercicios

En el caso del lenguaje Java, cuando la máquina virtual interpreta un programa, ejecuta un proceso. Una computadora con una sola CPU puede realizar multiprogramación al reasignar la CPU a varios procesos y así, éstos se ejecutan concurrentemente.
Dentro del proceso, el control puede seguir solamente un hilo de ejecución que por lo general comienza con el primer elemento del main, recorriendo una secuencia de instrucciones y terminando cuando se regresa al main. Un programa Java puede administrar varias secuencias de ejecución concurrentes. Cada secuencia es un hilo independiente y todos comparten tanto el espacio de direcciones como los recursos del sistema operativo. Por lo tanto, cada hilo puede acceder a todos los datos y procedimientos del programa, pero tiene su propio contador de programa y su pila de llamadas a procedimientos. Un hilo también se conoce como proceso ligero.

1. Estudiar cuáles métodos de la clase *Object* ayudan al procesamiento multihilos.
2. ¿Qué se entiende por exclusión mutua y por sincronización de procesos?
3. Diseñar un programa en el cual se crea y despacha un hilo hijo, de tal forma que el programa termina hasta que el hijo termina.
4. Escribir un programa concurrente para contar el número de enteros pares e impares presentes en un arreglo de números enteros.
5. Describir las diferencias entre un proceso UNIX-Linux y un hilo de Java, desde el punto de vista de la creación y administración de diferentes flujos de control, en un único programa concurrente escrito en C/C++ o en Java.
6. Elaborar un "manual de usuario" que explique el objeto de los siguientes métodos de la clase *Thread*: *checkAccess()*, *start()*, *stop()*, *isAlive()*, *suspend()*, *resume()*, *interrupt()*, *join()*, *destroy()*, *sleep()*, *yield()* y *run()*. En general, estudiar la clase *Thread*.
7. Escribir un programa de prueba que verifique el funcionamiento de los métodos enumerados en el ejercicio anterior.
8. Implementar el algoritmo de Peterson en lenguaje Java.
9. Escribir un programa que genere hilos, en particular demonios (daemon) con diversas tareas concurrentes entre sí. Elaborar dos versiones del mismo programa, generando los hilos como derivados de Thread o bien como una implementación de la interfaz *Runnable*.
10. Elaborar un diagrama de transición de estados de un hilo de Java, indicando qué métodos de las clases del ambiente de desarrollo, se corresponden con las transiciones del diagrama.

Apuntes


Introducción
Creación de Hilos
Secciones críticas
Exclusión mutua
Señalización de hilos
Bibliografía
Concurrencia en Java
75.59 - Técnicas de Programación Concurrente I
Facultad de Ingeniería - Universidad de Buenos Aires
75.59 - Técnicas de Programación Concurrente I
Concurrencia en Java


Introducción
Creación de Hilos
Secciones críticas
Exclusión mutua
Señalización de hilos
Bibliografía
Resumen
1 Introducción
2 Creación de Hilos
3 Secciones críticas
4 Exclusión mutua
5 Señalización de hilos
6 Bibliografía
75.59 - Técnicas de Programación Concurrente I
Concurrencia en Java

Introducción (I)

- Se implementa mediante *hilos* o *threads*, no procesos
- Los hilos se ejecutan dentro de procesos
  - Comparten recursos que fueron asignados por el sistema operativo
  - Comparten espacio de direcciones de memoria
- Los hilos comparten datos mediante variables

Creación de hilos (I)

- Los hilos se pueden crear de dos formas:
  - Heredando de la clase `Thread`
  - Implementando la interfaz `Runnable`
- ¿Cuándo se usa cada una?
  - Desición de diseño, depende de la estructura de clases de la aplicación
  - Al implementar la interfaz `Runnable` se puede extender otra clase

Creación de hilos (II)

- Ejemplo heredando de `Thread`

```
public class HiloThread extends Thread {

     public void run() {
          System.out.println ( "Hola, soy el hilo " +
              Thread.currentThread().getId() + " y
              heredo de Thread" );
          System.out.println ( "Termine" );
     }
}
```


Introducción
Creación de Hilos
Secciones críticas
Exclusión mutua
Señalización de hilos
Bibliografía

Creación de hilos (III)

- Ejemplo implementando `Runnable`

```
public class HiloRunnable implements Runnable {

     public void run() {
          System.out.println ( "Hola, soy el hilo " +
              Thread.currentThread().getId() + " e
              implemento Runnable" );
          System.out.println ( "Termine" );
     }
}
```

75.59 - Técnicas de Programación Concurrente I   Concurrencia en Java

Creación de hilos (III)

- Ejemplo: programa principal

```
public class Ejemplo1 {

     public static void main ( String[] args ) {
          HiloThread hilo1 = new HiloThread ();
          Thread hilo2 = new Thread ( new HiloRunnable
              () );

          hilo1.start ();
          hilo2.start ();
     }
}
```


Secciones críticas (I)

- Bloques *synchronized*: mecanismo propio de Java
- Dos partes:
  - Un objeto que servirá como lock
  - Un bloque de código a ejecutar en forma atómica
- Métodos *synchronized*: si un bloque de código es un método completo
- ¿Cómo funciona?
  - Cada objeto tiene un *lock* o *monitor*
  - Sólo un hilo a la vez puede tomar el lock

Secciones críticas (II)

- Ejemplo de bloque *synchronized*

```
public void incrementar ( int cantidad ) {
      synchronized ( this ) {
            this.valor += cantidad;
            System.out.println ( "Contador: valor actual
                = " + this.valor );
      }
}
```

- Ejemplo de método *synchronized*

```
public synchronized void incrementar ( int cantidad ) {
      this.valor += cantidad;
      System.out.println ( "Contador: valor actual = " +
            this.valor );
}
```


Introducción
Creación de Hilos
Secciones críticas
Exclusión mutua
Señalización de hilos
Bibliografía

Secciones críticas (III)

- Ejemplo de bloque *synchronized* en método estático

```
public static void escribirMensaje ( String mensaje ) {
      synchronized ( Contador.class ) {
            System.out.println ( "Mensaje del contador" )
                ;
      }
}
```

- Ejemplo de método estático *synchronized*

```
public static synchronized void escribirMensaje (
     String mensaje ) {
      System.out.println ( "Mensaje del contador" );
}
```

75.59 - Técnicas de Programación Concurrente I   Concurrencia en Java

Exclusión mutua (I)

- Los hilos participantes deben sincronizarse con el mismo objeto
- Ejemplo:

```
public static void main ( String[] args ) {
      Contador contador = new Contador ();
      Thread hilo1 = new Thread ( new Hilo(contador) );
      Thread hilo2 = new Thread ( new Hilo(contador) );

      hilo1.start ();
      hilo2.start ();
}
```


Introducción
Creación de Hilos
Secciones críticas
Exclusión mutua
Señalización de hilos
Bibliografía

Exclusión mutua (II)

- Ejemplo donde no hay exclusión mutua

```
public static void main ( String[] args ) {
      Contador contador1 = new Contador ();
      Contador contador2 = new Contador ();

      Thread hilo1 = new Thread ( new Hilo(contador1) );
      Thread hilo2 = new Thread ( new Hilo(contador2) );

      hilo1.start ();
      hilo2.star ();
}
```

75.59 - Técnicas de Programación Concurrente I   Concurrencia en Java

Señalización de hilos (I)

- Se debe tener el monitor adquirido para poder llamar a los siguientes métodos:
  - Método *wait()*: libera el monitor adquirido y suspende el hilo hasta que otro hijo llame a *notify()* o *notifyAll()*
  - Método *notify()*: despierta alguno de los hilos que espera por el monitor
  - Método *notifyAll()*: despierta todos los hilos que esperan por el monitor

Introducción
Creación de Hilos
Secciones críticas
Exclusión mutua
Señalización de hilos
Bibliografía

Señalización de hilos (II)

- Ejemplo de buffer con sincronismo:

```
public class Buffer {
      private int valor = 0;

      public synchronized int getValor () {
            try {
                  wait ();
            } catch ( InterruptedException e ) {}

            return valor;
      }

      public synchronized void setValor ( int valor ) {
            this.valor = valor;
            notifyAll ();
      }
}
```

75.59 - Técnicas de Programación Concurrente I   Concurrencia en Java

Variables volatile

- Los hilos guardan los valores de las variables compartidas en sus caches
- La palabra clave *volatile* indica al compilador que el valor de la variable no debe cachearse y debe leerse siempre de la memoria principal
- De este modo, los hilos verán siempre el valor más actualizado de la variable
- La declaración de una variable como *volatile* no realiza ningún lockeo en dicha variable

Bibliografía

- "Java Concurrency in Practice", Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes y Doug Lea
- Tutorial de concurrencia de Oracle, http://docs.oracle.com/javase/tutorial/essential/concurrency/

Fuentes de los ejemplos

Listado 1: Ejemplo 1: Main

```java
package org.tcp1.ejemplo1.main;

import org.tcp1.ejemplo1.hilos.HiloRunnable;
import org.tcp1.ejemplo1.hilos.HiloThread;

public class Ejemplo1 {

        public static void main ( String[] args ) {
                HiloThread hilo1 = new HiloThread ();
                Thread hilo2 = new Thread ( new HiloRunnable() );

                hilo1.start ();
                hilo2.start ();
        }

}
```

Listado 2: Ejemplo 1: Clase HiloRunnable

```java
package org.tcp1.ejemplo1.hilos;

public class HiloRunnable implements Runnable {

        @Override
        public void run () {
                System.out.println ( "Hola, soy el hilo " + Thread.currentThread().getId() + "
                     e implemento Runnable" );
                System.out.println ( "Termine" );
        }

}
```

Listado 3: Ejemplo 1: Clase HiloThread

```java
package org.tcp1.ejemplo1.hilos;

public class HiloThread extends Thread {

        public void run () {
                System.out.println ( "Hola, soy el hilo " + Thread.currentThread().getId() + "
                     y heredo de Thread" );
                System.out.println ( "Termine" );
        }
}
```

Listado 4: Ejemplo 2a: Main

```java
package org.tcp1.ejemplo2.main;

import org.tcp1.ejemplo2.contador.Contador;
import org.tcp1.ejemplo2.hilos.Hilo;

public class Ejemplo2a {

        public static void main ( String[] args ) {
                Contador contador = new Contador ();

                Thread hilo1 = new Thread ( new Hilo(contador) );
                Thread hilo2 = new Thread ( new Hilo(contador) );

                hilo1.start ();
                hilo2.start ();
        }

}
```

Listado 5: Ejemplo 2b: Main

```java
package org.tcp1.ejemplo2.main;

import org.tcp1.ejemplo2.contador.Contador;
import org.tcp1.ejemplo2.hilos.Hilo;
```

```java

public class Ejemplo2b {

        public static void main ( String[] args ) {
                Contador contador1 = new Contador ();
                Contador contador2 = new Contador ();

                Thread hilo1 = new Thread ( new Hilo(contador1) );
                Thread hilo2 = new Thread ( new Hilo(contador2) );

                hilo1.start ();
                hilo2.start ();
        }
}
```

Listado 6: Ejemplo 2: Clase Hilo

```java
package org.tcp1.ejemplo2.hilos;

import java.util.Random;

import org.tcp1.ejemplo2.contador.Contador;

public class Hilo implements Runnable {

        private Contador contador;

        public Hilo ( Contador contador ) {
                this.contador = contador;
        }

        @Override
        public void run () {
                for ( int i=0;i<10;i++ ) {
                        Random rand = new Random ();
                        int cantidad = rand.nextInt ( 100 );
                        System.out.println ( "Hilo " + Thread.currentThread().getId() + "
                             incrementa en " + cantidad + " el contador" );
                        contador.incrementar ( cantidad );
                }
        }

}
```

Listado 7: Ejemplo 2: Clase Contador

```java
package org.tcp1.ejemplo2.contador;

public class Contador {

        private int valor;

        public Contador () {
                valor = 0;
        }

        public int getValor() {
                return valor;
        }

        public void incrementar ( int cantidad ) {
                synchronized ( this ) {
                        this.valor += cantidad;
                        System.out.println ( "Contador: valor actual = " + this.valor );
                }
        }

        public static synchronized void escribirMensaje ( String mensaje ) {
                System.out.println ( "Mensaje del Contador" );
        }
}
```

Listado 8: Ejemplo 3: Main

```java
package org.tcp1.ejemplo3.main;

import org.tcp1.ejemplo3.buffer.Buffer;
import org.tcp1.ejemplo3.hilos.HiloEscritor;
import org.tcp1.ejemplo3.hilos.HiloLector;
```

```
public class Ejemplo3 {

	private static final int VUELTAS = 5;

	public static void main ( String[] args ) {
		Buffer buffer = new Buffer ();

		Thread hiloLector = new Thread ( new HiloLector(buffer,VUELTAS) );
		Thread hiloEscritor = new Thread ( new HiloEscritor(buffer,VUELTAS) );

		hiloLector.start ();
		hiloEscritor.start ();
	}

}
```

Listado 9: Ejemplo 3: Clase HiloLector

```
package org.tcp1.ejemplo3.hilos;

import org.tcp1.ejemplo3.buffer.Buffer;

public class HiloLector implements Runnable {

	private Buffer buffer;
	private int vueltas;

	public HiloLector ( Buffer buffer,int vueltas ) {
		this.buffer = buffer;
		this.vueltas = vueltas;
	}

	@Override
	public void run () {
		for ( int i=0;i<vueltas;i++ ) {
			System.out.println ( "HiloLector: esperando por el dato" );
			int dato = buffer.getValor ();
			System.out.println ( "HiloLector: dato leido del buffer = " + dato );
		}
	}

}
```

Listado 10: Ejemplo 3: Clase HiloEscritor

```
package org.tcp1.ejemplo3.hilos;

import java.util.Random;

import org.tcp1.ejemplo3.buffer.Buffer;

public class HiloEscritor implements Runnable {

	private Buffer buffer;
	private int vueltas;

	public HiloEscritor ( Buffer buffer,int vueltas ) {
		this.buffer = buffer;
		this.vueltas = vueltas;
	}

	@Override
	public void run () {
		for ( int i=0;i<vueltas;i++ ) {
			int aDormir = this.calcularRandom ( 10 );
			System.out.println ( "HiloEscritor: durmiendo " + aDormir + " segundos"
				);

			try {
				Thread.sleep ( 1000 * aDormir );
			} catch ( InterruptedException e ) {}

			int dato = this.calcularRandom ( 100 );
			System.out.println ( "HiloEscritor: escribiendo " + dato + " en el
				buffer" );
			buffer.setValor ( dato );
		}
	}

```

```
        private int calcularRandom ( int maximo ) {
                Random rand = new Random ();
                return rand.nextInt ( maximo );
        }

}
```

Listado 11: Ejemplo 3: Clase Buffer

```
package org.tcp1.ejemplo3.buffer;

public class Buffer {

        private int valor;

        public Buffer () {
                valor = 0;
        }

        public synchronized int getValor () {
                try {
                        wait ();
                } catch ( InterruptedException e ) {}
                return valor;
        }

        public synchronized void setValor ( int valor ) {
                this.valor = valor;
                notifyAll ();
        }

}
```

Listado 12: Ejemplo 4: Main

```
package org.tcp1.ejemplo4.main;

import org.tcp1.ejemplo4.buffer.Buffer;
import org.tcp1.ejemplo4.hilos.Consumidor;
import org.tcp1.ejemplo4.hilos.Productor;

public class Ejemplo4 {

        private static final int VUELTAS = 5;

        public static void main(String[] args) {

                Buffer buffer = new Buffer ();

                Thread productor = new Thread ( new Productor(buffer,VUELTAS) );
                Thread consumidor = new Thread ( new Consumidor(buffer,VUELTAS) );

                productor.start ();
                consumidor.start ();

                try {
                        productor.join ();
                        consumidor.join ();
                        System.out.println ( "*** Fin del programa" );
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

}
```

Listado 13: Ejemplo 4: Clase Consumidor

```
package org.tcp1.ejemplo4.hilos;

import org.tcp1.ejemplo4.buffer.Buffer;

public class Consumidor implements Runnable {

        private Buffer buffer;
        private int vueltas;

        public Consumidor ( Buffer b,int vueltas ) {
                this.buffer = b;
```

```
                this.vueltas = vueltas;
        }

        @Override
        public void run() {

                for ( int i=0;i<vueltas;i++ ) {

                        // calcular valor random a dormir
                        int aDormir = calcularRandom ( 1,10 );

                        // leer el valor del buffer
                        int valor = buffer.getValor ();
                        System.out.println ( "Consumidor: valor leido del buffer = " + valor +
                            " (tiempo a dormir = " + aDormir + ")" );

                        try {
                                Thread.sleep ( aDormir*1000 );
                        } catch (InterruptedException e) {}
                }
        }

        private int calcularRandom ( int max,int min ) {
                return (int)(Math.random()*(max-min))+min;
        }
}
```

Listado 14: Ejemplo 4: Clase Productor

```
package org.tcp1.ejemplo4.hilos;

import org.tcp1.ejemplo4.buffer.Buffer;

public class Productor implements Runnable {

        private Buffer buffer;
        private int vueltas;

        public Productor ( Buffer b,int vueltas ) {
                this.buffer = b;
                this.vueltas = vueltas;
        }

        @Override
        public void run() {

                for ( int i=0;i<vueltas;i++ ) {

                        // calcular valor random para guardar en el buffer
                        int aGuardar = calcularRandom ( 1,100 );

                        // calcular valor random a dormir
                        int aDormir = calcularRandom ( 1,10 );

                        buffer.setValor ( aGuardar );
                        System.out.println ( "Productor: valor guardado en el buffer = " +
                            aGuardar + " (tiempo a dormir = " + aDormir + ")" );

                        try {
                                Thread.sleep ( aDormir*1000 );
                        } catch (InterruptedException e) {}

                }
        }

        private int calcularRandom ( int max,int min ) {
                return (int)(Math.random()*(max-min))+min;
        }
}
```

Listado 15: Ejemplo 4: Clase Buffer

```
package org.tcp1.ejemplo4.buffer;

public class Buffer {

        private int valor;
        private boolean disponible;

        public Buffer () {
```

```
                valor = 0;
                disponible = false;
        }

        public synchronized int getValor () {

                // el consumidor toma el monitor
                while ( disponible == false ) {
                        try {
                                wait ();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }

                disponible = false;
                notifyAll ();

                return valor;
                // el consumidor libera el monitor
        }

        public synchronized void setValor ( int v ) {

                // el productor toma el monitor
                while ( disponible == true ) {
                        try {
                                wait ();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }

                valor = v;
                disponible = true;
                notifyAll ();
                // el productor libera el monitor
        }
}
```

Listado 16: Ejemplo 5: Main

```
package org.tcp1.ejemplo5.main;

import org.tcp1.ejemplo5.buffer.Buffer;
import org.tcp1.ejemplo5.buffer.Semaforo;
import org.tcp1.ejemplo5.hilos.Consumidor;
import org.tcp1.ejemplo5.hilos.Productor;

public class Ejemplo5 {

        private static final int VUELTAS = 5;

        public static void main ( String[] args ) {

                Buffer buffer = new Buffer ();
                Semaforo semaforoLectura = new Semaforo ( 0 );
                Semaforo semaforoEscritura = new Semaforo ( 1 );

                Thread thrProductor = new Thread ( new Productor(buffer,semaforoLectura,
                     semaforoEscritura,VUELTAS) );
                Thread thrConsumidor = new Thread ( new Consumidor(buffer,semaforoLectura,
                     semaforoEscritura,VUELTAS) );

                thrProductor.start ();
                thrConsumidor.start ();

                try {
                        thrProductor.join ();
                        thrConsumidor.join ();
                        System.out.println ( "*** Fin del programa" );
                } catch ( InterruptedException e ) {
                        e.printStackTrace();
                }
        }

}
```

Listado 17: Ejemplo 5: Clase Consumidor

```java
package org.tcp1.ejemplo5.hilos;

import org.tcp1.ejemplo5.buffer.Buffer;
import org.tcp1.ejemplo5.buffer.Semaforo;

public class Consumidor implements Runnable {

        private Buffer buffer;
        private Semaforo semaforoLectura;
        private Semaforo semaforoEscritura;
        private int vueltas;

        public Consumidor ( Buffer buffer,Semaforo semaforoLectura,Semaforo semaforoEscritura, int vueltas ) {
                this.buffer = buffer;
                this.semaforoLectura = semaforoLectura;
                this.semaforoEscritura = semaforoEscritura;
                this.vueltas = vueltas;
        }

        @Override
        public void run() {

                for ( int i=0;i<vueltas;i++ ) {

                        // calcular valor random a dormir
                        int aDormir = calcularRandom ( 1,10 );

                        // el consumidor pide permiso para leer el valor
                        System.out.println ( "Consumidor: esperando para leer" );
                        semaforoLectura.p ();
                        // leer el valor del buffer
                        int valor = buffer.getValor ();
                        System.out.println ( "Consumidor: valor leido del buffer = " + valor + " (tiempo a dormir = " + aDormir + ")" );
                        // el consumidor notifica al productor que puede escribir el siguiente valor
                        System.out.println ( "Consumidor: notificando al productor" );
                        semaforoEscritura.v ();

                        try {
                                Thread.sleep ( aDormir*1000 );
                        } catch (InterruptedException e) {}
                }
        }

        private int calcularRandom ( int max,int min ) {
                return (int)(Math.random()*(max-min))+min;
        }
}
```

Listado 18: Ejemplo 5: Clase Productor

```java
package org.tcp1.ejemplo5.hilos;

import org.tcp1.ejemplo5.buffer.Buffer;
import org.tcp1.ejemplo5.buffer.Semaforo;

public class Productor implements Runnable {

        private Buffer buffer;
        private Semaforo semaforoLectura;
        private Semaforo semaforoEscritura;
        private int vueltas;


        public Productor ( Buffer buffer,Semaforo semaforoLectura,Semaforo semaforoEscritura, int vueltas ) {
                this.buffer = buffer;
                this.semaforoLectura = semaforoLectura;
                this.semaforoEscritura = semaforoEscritura;
                this.vueltas = vueltas;
        }

        @Override
        public void run() {

                for ( int i=0;i<vueltas;i++ ) {

                        // calcular valor random para guardar en el buffer
                        int aGuardar = calcularRandom ( 1,100 );
```

```java

            // calcular valor random a dormir
            int aDormir = calcularRandom ( 1,10 );

            // el productor pide permiso para escribir en el buffer
            System.out.println ( "Productor: esperando para escribir" );
            semaforoEscritura.p ();
            buffer.setValor ( aGuardar );
            System.out.println ( "Productor: valor guardado en el buffer = " +
                aGuardar + " (tiempo a dormir = " + aDormir + ")" );
            // el productor notifica al consumidor que puede leer
            System.out.println ( "Productor: notificando al consumidor" );
            semaforoLectura.v ();

            try {
                Thread.sleep ( aDormir*1000 );
            } catch (InterruptedException e) {}

        }
    }

    private int calcularRandom ( int max,int min ) {
        return (int)(Math.random()*(max-min))+min;
    }
}
```

Listado 19: Ejemplo 5: Clase Buffer

```java
package org.tcp1.ejemplo5.buffer;

public class Buffer {

    private int valor;

    public Buffer () {
        valor = 0;
    }

    public int getValor() {
        return valor;
    }

    public void setValor(int valor) {
        this.valor = valor;
    }

}
```

Listado 20: Ejemplo 5: Clase Semaforo

```java
package org.tcp1.ejemplo5.buffer;

public class Semaforo {

    private int valor;

    public Semaforo ( int valorInicial ) {
        valor = valorInicial;
    }

    /**
     * Resta una unidad al valor del semaforo
     */
    public synchronized void p () {

        // si el valor del semaforo es 0 => el hilo debe bloquearse
        while ( valor == 0 )
            try {
                // el hilo libera el monitor y se bloquea
                wait ();
            } catch ( InterruptedException e ) {
                e.printStackTrace();
            }

        // cuando el valor no es cero => se resta una unidad
        valor --;
    }

    /**
     * Suma una unidad al valor del semaforo
```

```
         */
        public synchronized void v () {
                // se suma una unidad al semaforo y se notifica a los hilos que estaban
                     bloqueados
                valor ++;
                notifyAll ();
        }
}
```

Listado 21: Ejemplo 6: Main

```
package org.tcp1.ejemplo6;

import java.util.Random;

public class Main {

        private static final int CANTIDAD_ITEMS = 4;

        private static Random generadorRandom;

        public static void main ( String[] args ) {

                float buffer[] = new float[CANTIDAD_ITEMS];

                // se inicializan los valores iniciales
                generadorRandom = new Random ();
                for ( int i=0;i<CANTIDAD_ITEMS;i++ )
                        buffer[i] = generadorRandom.nextFloat ();

                Coordinador coordinador = new Coordinador ( buffer,CANTIDAD_ITEMS );
                coordinador.procesar ();

                System.out.println ( "Fin del programa" );
        }

}
```

Listado 22: Ejemplo 6: Clase Coordinador

```
package org.tcp1.ejemplo6;

import java.util.concurrent.CyclicBarrier;

public class Coordinador {

        private static final int CANTIDAD_VUELTAS = 5;

        private float buffer[];
        private int cantidad;
        private CyclicBarrier barrera;

        public Coordinador ( float[] bufferInicial,int cantidad ) {
                this.barrera = new CyclicBarrier ( cantidad );
                this.buffer = bufferInicial;
                this.cantidad = cantidad;
        }

        public void procesar () {

                Thread workers[] = new Thread[this.cantidad];

                // se lanzan los hilos
                for ( int i=0;i<this.cantidad;i++ ) {
                        workers[i] = new Thread ( new Worker(i,buffer,cantidad,barrera,
                             CANTIDAD_VUELTAS) );
                        workers[i].start ();
                }

                // se espera a que terminen los hilos
                for ( int i=0;i<this.cantidad;i++ ) {
                        try {
                                workers[i].join ();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }

                System.out.println ( "Coordinador: se termino el trabajo" );
        }
```

```
}
```

Listado 23: Ejemplo 6: Clase Worker

```
package org.tcp1.ejemplo6;

import java.util.Arrays;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;

public class Worker implements Runnable {

        private int posicionDelBuffer;
        private float[] buffer;
        private int cantidad;
        private CyclicBarrier barrera;
        private int vueltas;

        public Worker ( int posicion,float[] bufferInicial,int cantidad,CyclicBarrier barrera,
             int vueltas ) {
                this.posicionDelBuffer = posicion;
                this.buffer = bufferInicial;
                this.cantidad = cantidad;
                this.barrera = barrera;
                this.vueltas = vueltas;
        }

        @Override
        public void run () {
                for ( int i=0;i<this.vueltas;i++ ) {

                        // se obtiene el resultado del calculo
                        float resultado = procesarBuffer ();

                        try {
                                // se imprimen los datos en pantalla
                                System.out.println ( "Vuelta " + (i+1) + " - Worker " + this.
                                     posicionDelBuffer + " - Buffer inicial " + Arrays.toString(
                                     this.buffer) + " - Resultado = " + resultado );

                                // se espera a que todos terminen el calculo antes de
                                     actualizar el buffer
                                this.barrera.await ();

                                // se guarda el valor obtenido en la posicion correspondiente
                                     para la siguiente vuelta
                                this.buffer[this.posicionDelBuffer] = resultado;

                                // se espera a que todos terminen de escribir en el buffer
                                     antes de iniciar la siguiente vuelta
                                this.barrera.await ();

                        } catch ( InterruptedException e ) {
                                e.printStackTrace();
                        } catch ( BrokenBarrierException e ) {
                                e.printStackTrace ();
                        }
                }
        }

        private float procesarBuffer () {
                float resultado = 0;
                for ( int i=0;i<this.cantidad;i++ )
                        resultado += this.buffer[i];

                return resultado;
        }

}
```